propagation_uncertainty_cov.R

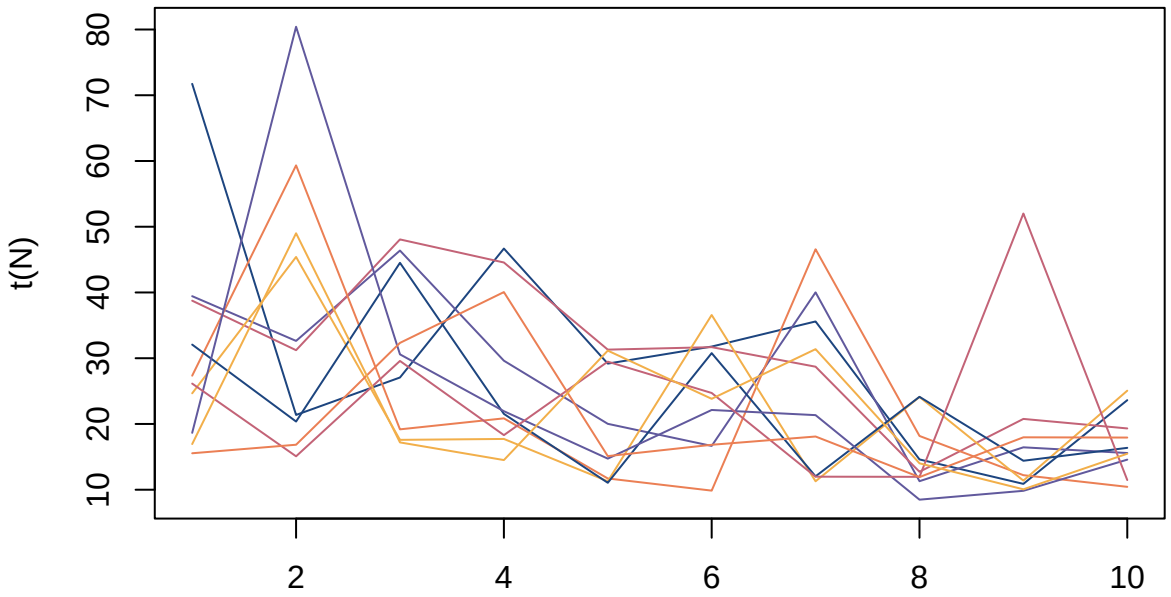
katherine

2022-06-29

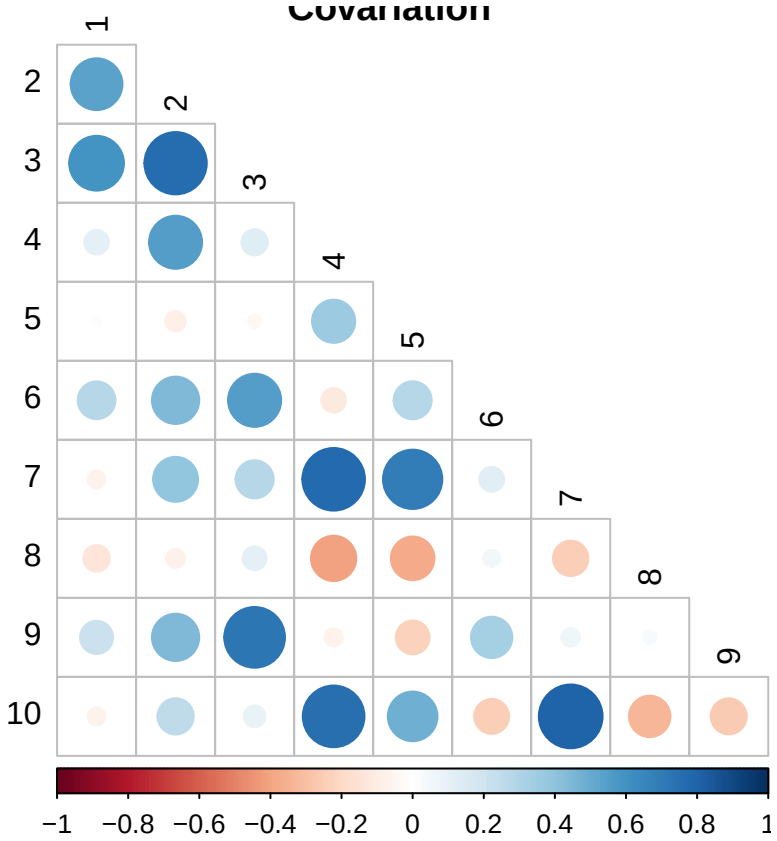
```
# Demo the propagation of uncertainty in the calculation of the LPI

library(corrplot)

## simulate populations ------------------------------------------------------

# Function to simulate populations
# For this demo, I am not adding any interactions between populations

sigma_process = 0.4
sigma_measure = 0.4
N0 = 40
n_pops = 5
interaction_effect = 0.2


## carrying capacity scenarios ----

K_increase = N0 + 5*c(0:10)
K_stable = rep(N0, 11)
K_decline = N0 - 2.5*c(0:10)

K = K_decline
lpiylim = c(0, 4)

sim_mech <- function(
    n_pairs = 10,
    timesteps = 10,
    N0i = N0, N0j = N0,
    lambda_i = 1.5, lambda_j = 1.5,
    alpha_ij = interaction_effect, alpha_ji = interaction_effect, # covariance
    process = proc,
    observation = obs,
    K){

  # set seed for randomisations
  set.seed(2)

  ## arguments: ## ----

  # npairs = number of pairs of populations to simulate
  # timesteps = number of time steps
```

```r
# N0i = initial population size for set i
# N0j = initial population size for set j
# lambda_i = true population growth rate for set i
# lambda_j = true population growth rate for set j
# alpha_ij = interaction coefficient (effect of set j on set i)
# alpha_ji = interaction coefficient (effect of set i on set j)
# K = carrying capacity of the environment
# process = absolute value of the  to add to growth rates
# observation = absolute value of the obs error limit to add to population sizes

# add more time steps to allow lag
timesteps = timesteps

# initialize matrix to store results (population sizes)
Ni <- as.matrix(rep(N0i, n_pairs))
Nj <- as.matrix(rep(N0j, n_pairs))

# initialize matrix to store growth rates
dt_i <- matrix(NA, nrow = n_pairs, ncol = timesteps)
dt_j <- matrix(NA, nrow = n_pairs, ncol = timesteps)

# Nt+1_i = Nt_i + rNt_i * ((1 - Nt_i/K_i) + alpha_ji*Nt_j/K_j
# calculate population sizes
for(t in 1:timesteps){

  # generate growth rates from a lognormal distribution with process error
  r_i_error = rlnorm(n = n_pairs, meanlog = log(lambda_i), sdlog = process)
  r_j_error = rlnorm(n = n_pairs, meanlog = log(lambda_j), sdlog = process)

  # population i
  temp_i = Ni[,t]*(1 + r_i_error*(1 - (Ni[,t] + alpha_ij*Nj[,t])/K[t]))

  # if NAs or 0, set to 0.
  temp_i[which(is.na(temp_i))] <- 0
  temp_i[which(temp_i < 0)] <- 0
  Ni <- cbind(Ni, temp_i) # append resulting population size to results vector

  # population j
  temp_j = Nj[,t]*(1 + r_j_error*(1 - (Nj[,t] + alpha_ji*Ni[,t])/K[t]))

  # if NAs or 0, set to 0.
  temp_j[which(is.na(temp_j))] <- 0
  temp_j[which(temp_j < 0)] <- 0
  Nj <- cbind(Nj, temp_j) # append resulting population size to results vector

  ## save generated growth rates
  dt_i[,t] <- r_i_error
  dt_j[,t] <- r_j_error
}

# apply observation error on the calculated population sizes
# pick number of lognormal distribution with a mean of N and an sd of observation error
Ni <- apply(Ni, 1:2, function(x) {rlnorm(1, meanlog = log(x), sdlog = observation)})
```

```
  Nj <- apply(Nj, 1:2, function(x) {rlnorm(1, meanlog = log(x), sdlog = observation)})

  return(list(Ni, Nj, dt_i, dt_j))
}


## simulate ----

N_all <- sim_mech(n_pairs = n_pops, timesteps = 10,
                  process = sigma_process, observation = sigma_measure,
                  K = K) # change direction of change ----
N <- rbind(N_all[[1]][,-1], N_all[[2]][,-1])
matplot(t(N), type = "l", col = PNWColors::pnw_palette("Sunset2", 5), lty= 1)
```

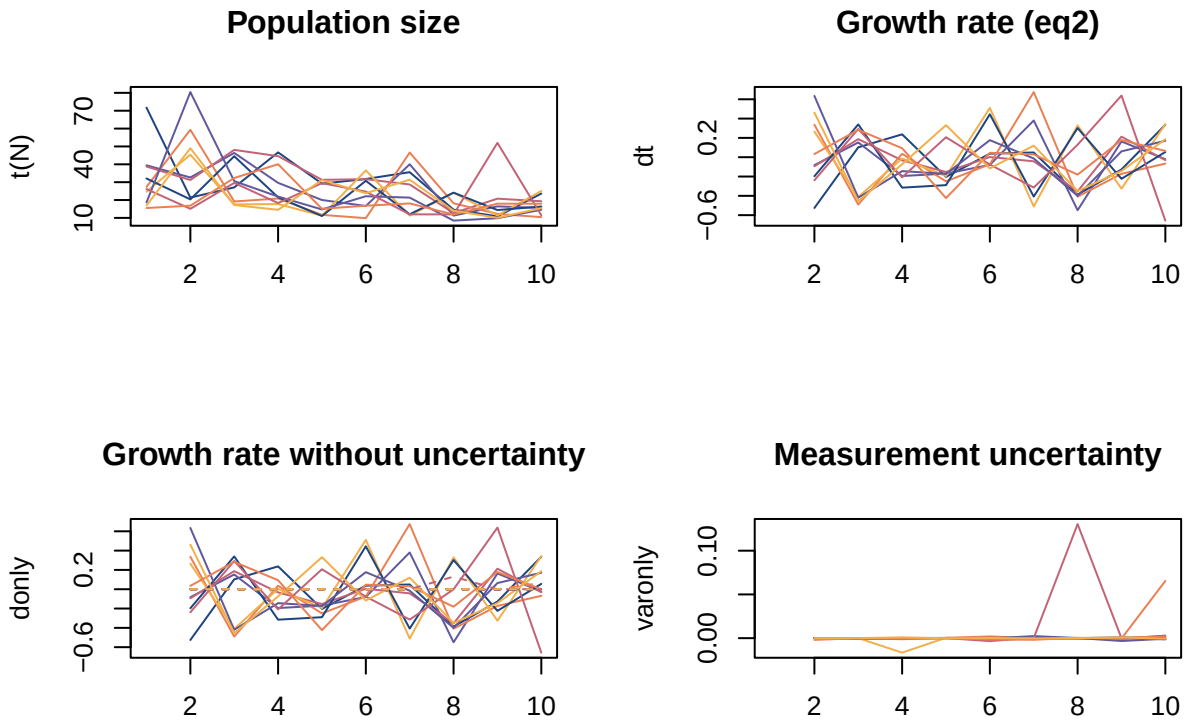



```
## check the covariance between populations

cov(t(N)) %>% cov2cor() %>% corrplot(title = "Covariation", tl.col = "black", type = "lower", diag = FA
```



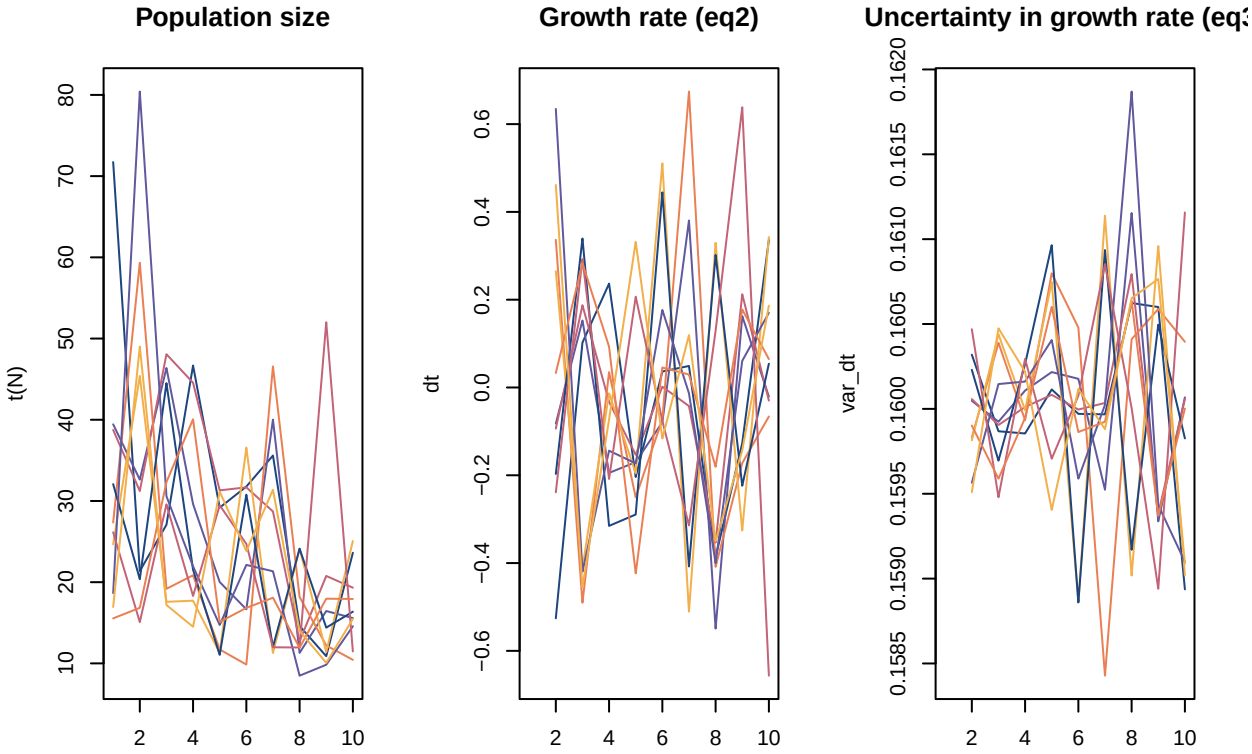

```
#### estimate population growth rate trend #### ------------------------------

# equation 2 ----
# expectation of the growth rate trend
eq2 <- function(N){
  d <- c()
  for(t in 2:length(N)){
    d[t] <- log10(N[t]/N[t-1]) + (sigma_measure^2)/(2*(N[t-1]^2 - N[t]^2))
  }
  return(d)
}

dt <- apply(N, 1, eq2)

# equation 2
# just the growth rate part
eq2_donly <- function(N){
  eq2dt <- c()
  for(t in 2:length(N)){
    eq2dt[t] <- log10(N[t]/N[t-1])
  }
  return(eq2dt)
}
donly <- apply(N, 1, eq2_donly)


# equation 2
```

```r
# just the uncertainty part
eq2_varonly <- function(N){
  eq2var <- c()
  for(t in 2:length(N)){
    eq2var[t] <- (sigma_measure^2)/(2*((N[t-1])^2 - (N[t])^2))
    # should the population sizes be logged here?
    #eq2var[t] <- (sigma_measure^2)/(2*(log(N[t-1])^2 - log(N[t])^2))
  }
  return(eq2var)
}
varonly <- apply(N, 1, eq2_varonly)

par(mfrow = c(2,2))
matplot(t(N), type = "l", main = "Population size", col = PNWColors::pnw_palette("Sunset2", 5), lty= 1)
matplot(dt, type = "l", main = "Growth rate (eq2)", col = PNWColors::pnw_palette("Sunset2", 5), lty= 1)
matplot(donly, type = "l", main = "Growth rate without uncertainty", col = PNWColors::pnw_palette("Suns
matplot(varonly, type = "l", main = "Measurement uncertainty", add = TRUE, col = PNWColors::pnw_palette
matplot(varonly, type = "l", main = "Measurement uncertainty", col = PNWColors::pnw_palette("Sunset2", 5
```

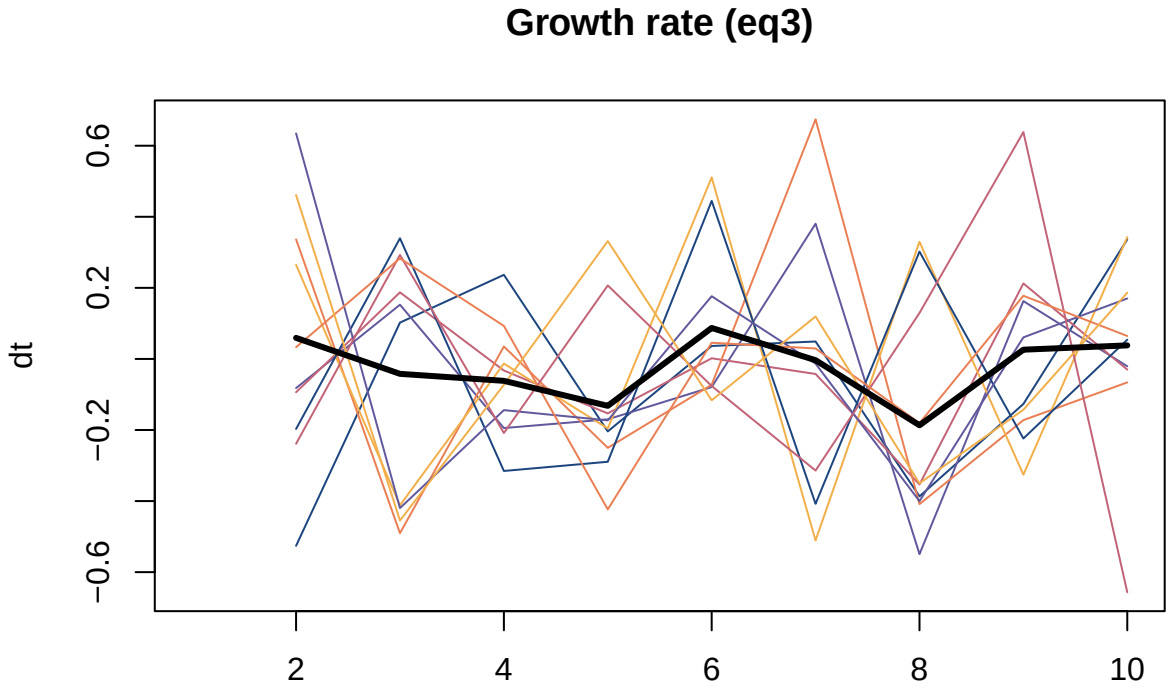



```r
## equation 3 ----

eq3 <- function(N){
  eq3var <- c()
  for(t in 2:length(N)){
    eq3var[t] <- sigma_process^2 + (sigma_measure^2)*((1/((N[t])^2) - (1/(N[t-1])^2)))
  }
  return(eq3var)
}
var_dt <- apply(N, 1, eq3)

par(mfrow = c(1,3))
```

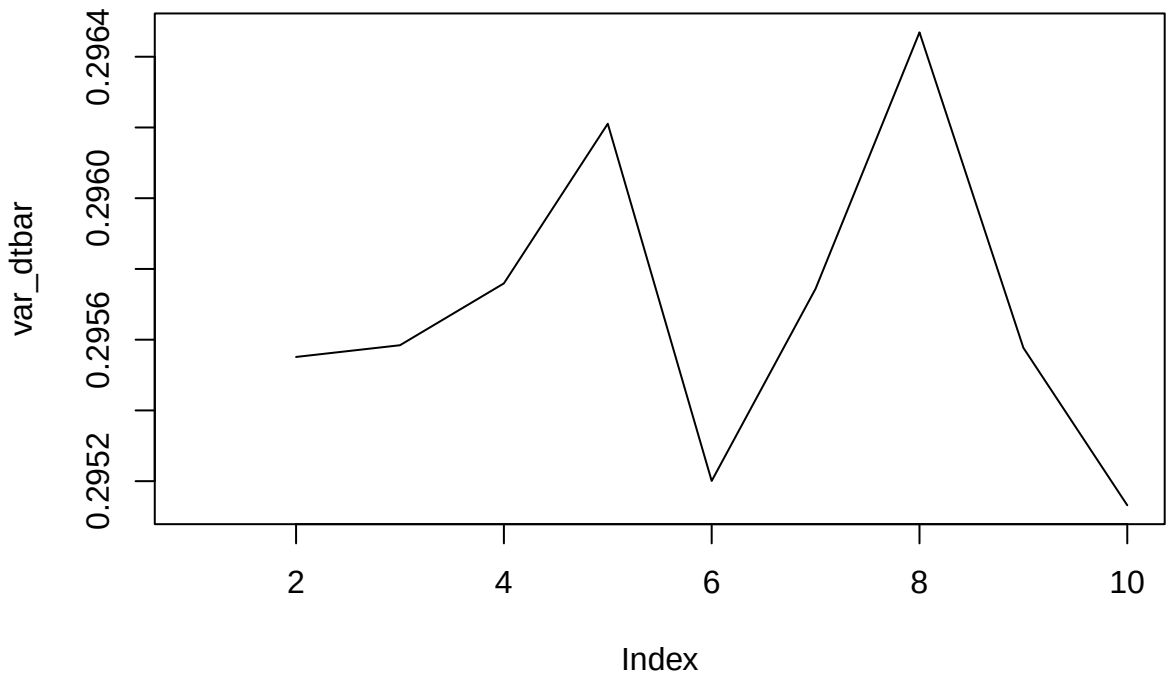
```
matplot(t(N), type = "l", main = "Population size", col = PNWColors::pnw_palette("Sunset2", 5), lty= 1)
matplot(dt, type = "l", main = "Growth rate (eq2)", col = PNWColors::pnw_palette("Sunset2", 5), lty= 1)
matplot(var_dt, type = "l", main = "Uncertainty in growth rate (eq3)", col = PNWColors::pnw_palette("Su
```



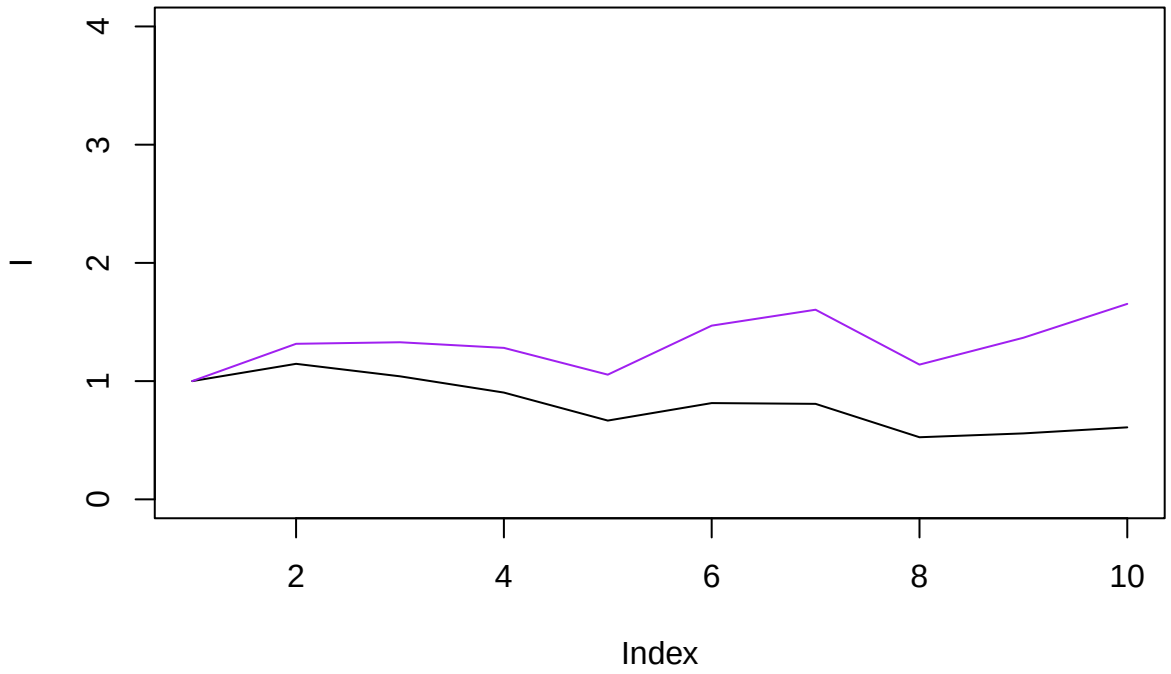

```
#### average growth rate #### ----------------------------------------------------

## equation 4 ----
## average growth rate at each time step

dt_bar <- apply(dt, 1, mean)

par(mfrow = c(1,1))
matplot(dt, type = "l", main = "Growth rate (eq3)", col = PNWColors::pnw_palette("Sunset2", 5), lty= 1)
lines(dt_bar, lwd = 3)
```




```
## equation 5 ----
## uncertainty in the average growth rate

dt_cov <- cov(dt, use = "pairwise.complete.obs")
dt_cov <- dt_cov[which(lower.tri(dt_cov))]

var_dtbar = (1/n_pops)*(apply(var_dt, 1, sum) + 2*sum(dt_cov))

plot(var_dtbar, type = "l")
```

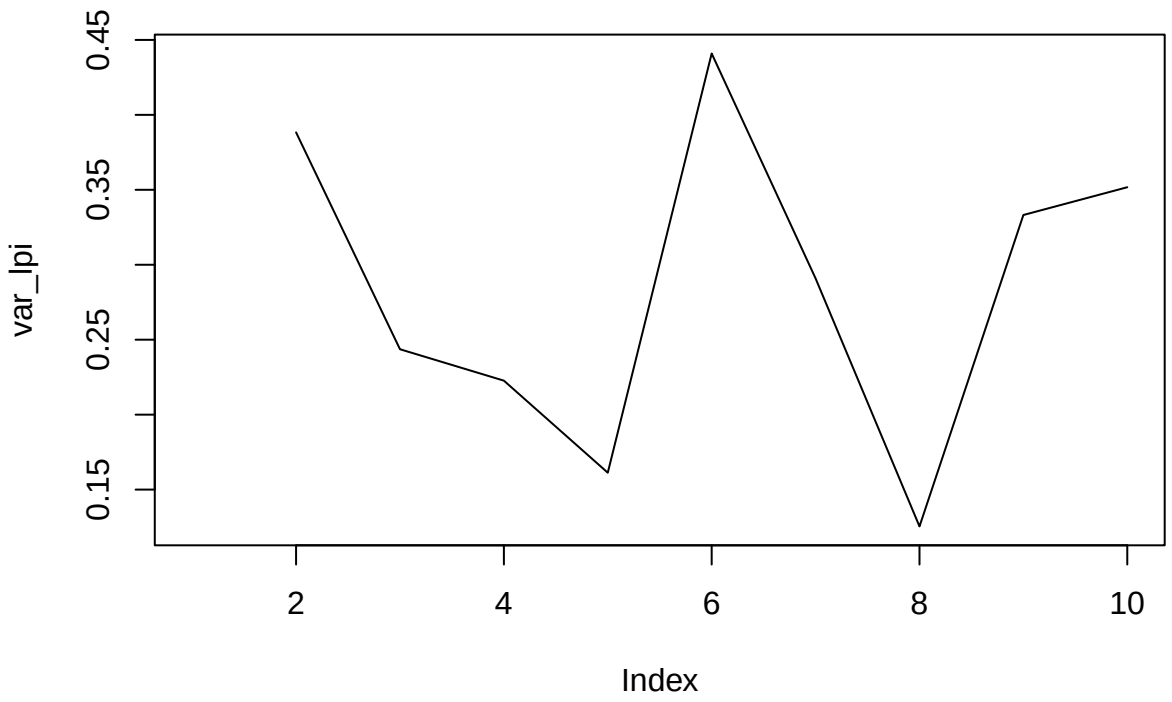


```r
## equation 6 ----
## calculate LPI (without uncertainty correction)

# function to calculate LPI value without uncertainty correction
calclpi <- function(dt_bar){
  I = 1
  for(i in 2:length(dt_bar)){
    I[i] <- I[i-1]*10^dt_bar[i]
  }
  return(I)
}

## equation 7 ----

# function to calculate LPI value WITH uncertainty correction
calclpi_corrected <- function(dt_bar){
  I = 1
  for(i in 2:length(dt_bar)){
    I[i] <- I[i-1]*10^dt_bar[i] + 0.5*(10^dt_bar[i]*var_dtbar[i])
  }
  return(I)
}

lpi_nocorrection = calclpi(dt_bar)
lpi_correction = calclpi_corrected(dt_bar)

plot(lpi_nocorrection, type = "l", ylab = "I", ylim = lpiylim)
lines(lpi_correction, col = "purple")
```




```r
## equation 8 ----

# function to obtain the variance of the LPI
```

```r
var_lpi <- (10^(2*dt_bar))*var_dtbar
plot(var_lpi, type = "l")
```




```r
# render
# rmarkdown::render("propagation_uncertainty_cov.R", output_format = "pdf_document", output_file = "pro
# rmarkdown::render("propagation_uncertainty_cov.R", output_format = "pdf_document", output_file = "pro
# rmarkdown::render("propagation_uncertainty_cov.R", output_format = "pdf_document", output_file = "pro
```